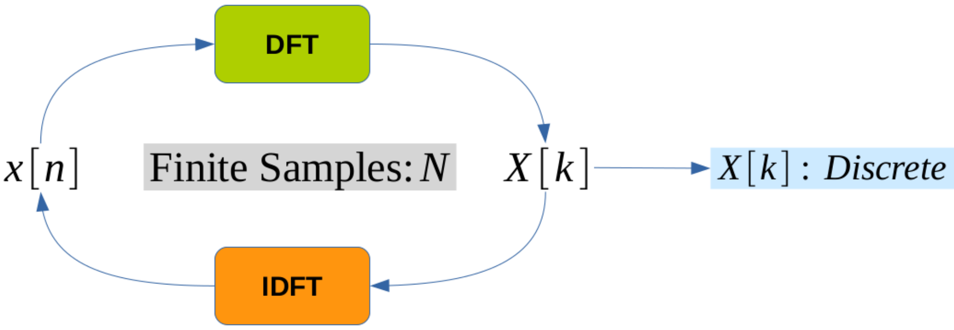
Experiment No : 7

Title : DFTs / IDFTs using matrix multiplication and also using commands

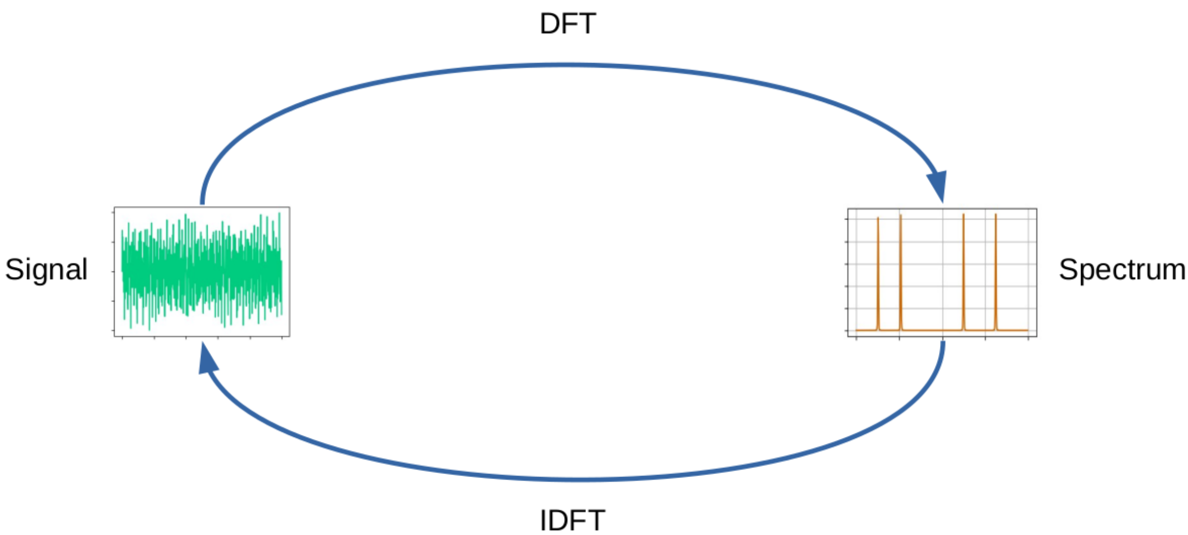
Objective : 1. Study the discrete spectrum of discrete time signal
2. Generate the discrete time signal from spectrum

Block Diagram :

$$X[k]=\sum_{n=0}^{N-1} x[n]e^{-j2\pi kn/N};\ 0\le k<N-1$$ *Analysis Equation*



$$x[n]=\frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j2\pi kn/N};\ 0\le n\le N-1$$ *Synthesis Equation*

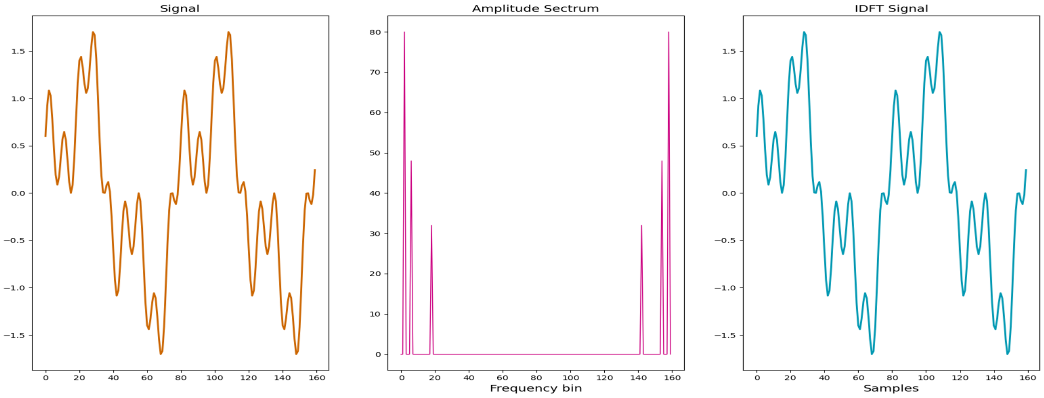


Code :

```
import numpy as np
import matplotlib.pyplot as plt
import dspmodule_custom as dsp
```

```
#window_flag = 0/1/2/3 [rec, bartleett, hamm, hann]
window_flag = 0
N = 160
n = np.arange(0,N)
x = np.sin(np.pi*n/40) + 0.6*np.cos(3*np.pi*n/40) + 0.4*np.sin(9*np.pi*n/40)

# Adjust spectrum resolution
N_dft = N

#Set N_dft = 256, with and with out hamming window and observe the spectrum

#N_dft = 256
X = dsp.dft(x, N= N_dft, win = window_flag)
print('[=] {} point DFT successfully computed.'.format(X.size))

x_hat = dsp.idft(X)
print('[=] {} point IDFT successfuly computed.'.format(x.size))

plt.figure(figsize=(20,4))
plt.subplot(1,3,1)
plt.plot(x,lw=2,color = [0.8,0.4,0])
plt.title('Signal',fontsize = 14)
plt.xlabel('Samples', fontsize = 14)
plt.subplot(1,3,2)
plt.plot(np.absolute(X), lw = 1, color = [0.8,0,0.5])
plt.title('Amplitude Sectrum',fontsize = 14)
plt.xlabel('Frequency bin',fontsize = 14)
plt.subplot(1,3,3)
plt.plot(x_hat.real, lw = 2, color = [0,0.6,0.7])
plt.title('IDFT Signal', fontsize = 14)
plt.xlabel('Samples', fontsize = 14)
plt.show()
```

Output :



Custom Module :

```
import numpy as np

def dft(signal, N=None, win=0):
"""
Compute the Discrete Fourier Transform (DFT) of a signal.

Parameters:
signal (numpy.ndarray): Input signal (1-D numpy array).
N (int or None): DFT point (optional). If None, N is set to the length of the signal.
win (int): Window type (0 for rectangular, 1 for Bartlett, 2 for Hamming, 3 for
Hann).

Returns:
numpy.ndarray: DFT spectrum (1-D numpy array).
"""
# Check if N is provided, otherwise use the length of the signal
if N is None:
N = len(signal)

# Generate the window based on the specified type
window = generate_window(N, win)

# Pad the signal with zeros to match the DFT point
if len(signal) < N:
signal = np.pad(signal, (0, N - len(signal)), 'constant')

# Compute the DFT using FFT
spectrum = np.fft.fft(signal * window, N)

return spectrum

def idft(spectrum):
"""
Compute the Inverse Discrete Fourier Transform (IDFT) of a spectrum.

Parameters:
spectrum (numpy.ndarray): Spectrum (1-D numpy array).

Returns:
numpy.ndarray: IDFT signal (1-D numpy array).
"""
# Compute the IDFT using the inverse FFT
signal = np.fft.ifft(spectrum)
```

```
return signal

def generate_window(N, win_type):
"""
Generate a window of a specified type and length.

Parameters:
N (int): Length of the window.
win_type (int): Window type (0 for rectangular, 1 for Bartlett, 2 for Hamming, 3
for Hann).

Returns:
numpy.ndarray: Window (1-D numpy array).
"""
if win_type == 0: # Rectangular window
window = np.ones(N)
elif win_type == 1: # Bartlett window
window = np.bartlett(N)
elif win_type == 2: # Hamming window
window = np.hamming(N)
elif win_type == 3: # Hann window
window = np.hanning(N)
else: # Default to rectangular window
window = np.ones(N)

return window

if __name__ == "__main__":
# You can add some test cases or examples here
pass
```